Q1

e) Explain in a few sentences what happens when the model's `train_op` is called (what gets computed during forward propagation, what gets computed during backpropagation, and what will have changed after the op has been run?).

The loss and the gradients of all tf_Variable()'s are computed in forward prop and back prop , respectively. The tf_Variable()'s are updated after each iteration.

Q2

a)

Iteration	Stack	Buffer	New Dependency	Transition
0	ROOT	I parsed this sentence correctly		
1	ROOT I	Parsed this sentence correctly		Shift
2	ROOT I parsed	This sentence correctly		Shift
3	ROOT parsed	This sentence correctly	Parsed -> I	Left-Arc
4	ROOT parsed this	Sentence correctly		Shift
5	ROOT parsed this sentence	correctly		Shift
6	ROOT parsed sentence	Correctly	Sentence -> this	Left-arc
7	ROOT parsed	Correctly	Parsed -> sentence	Right-arc
8	ROOT parsed correctly			Shift
9	ROOT parsed		Parsed -> correctly	Right-arc
10	ROOT		ROOT -> parsed	Right-arc

b) A sentence containing *n* words will be parsed in how many steps (in terms of *n*)? Briefly explain why.

A sentence with *n* words will be parsed in *2n* steps, since each word needs a SHIFT and an ARC step.

d) When implementing the mini-batch parser, shallow copies of the Parser list should be created. That way, when the self.dependencies of the unfinished Parser list are updated, the same is done for the original Parser list. Also, removing completed Parser objects in the unfinished Parser list will not affect the original this way.

Also, to remove completed Parsers in the unfinished Parser list, first create a shallow copy of a list of the first batch_size Parser objects, so that removing the completed Parser objects will not trigger an IndexError.

Use parse_step instead of parse, since parse performs multiple parsing with a list of transitions. If you pass in only one transition, it treats that transition as a list which means each parse_step in parse takes in a character from the transition. If you used an else for the last transition command (S, LA, RA), the parser will perform the last transition for each character that it misinterprets as a transition.

f)

(2 points, written) We will regularize our network by applying Dropout[3]. During training this randomly sets units in the hidden layer $\boldsymbol{h}$ to zero with probability p_{drop} and then multiplies $\boldsymbol{h}$ by a constant γ (dropping different units each minibatch). We can write this as

$$\boldsymbol{h}_{drop} = \gamma \boldsymbol{d} \circ \boldsymbol{h}$$

where $\boldsymbol{d} \in \{0, 1\}^{D_h}$ (D_h is the size of $\boldsymbol{h}$) is a mask vector where each entry is 0 with probability p_{drop} and 1 with probability $(1 - p_{drop})$. γ is chosen such that the value of $\boldsymbol{h}_{drop}$ in expectation equals $\boldsymbol{h}$:

$$\mathbb{E}_{p_{drop}}[\boldsymbol{h}_{drop}]_i = h_i$$

for all $0 < i < D_h$. What must γ equal in terms of p_{drop}? Briefly justify your answer.

$$\begin{aligned} E_{p_{drop}}[h_{drop}] &= \sum p_{drop} h_{drop} \\ &= p(h_{drop} = 0)(h_{drop} = 0) + p(h_{drop} = \gamma h)(h_{drop} = \gamma h) \\ &= p_{drop}(0) + (1 - p_{drop})(\gamma h) \end{aligned}$$

$$\therefore \gamma = \frac{1}{1 - p_{drop}}$$

g)

(i) First, Adam uses a trick called *momentum* by keeping track of $\boldsymbol{m}$, a rolling average of the gradients:

$$\begin{aligned} \boldsymbol{m} &\leftarrow \beta_1 \boldsymbol{m} + (1 - \beta_1) \nabla_{\boldsymbol{\theta}} J_{minibatch}(\boldsymbol{\theta}) \\ \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} - \alpha \boldsymbol{m} \end{aligned}$$

where β_1 is a hyperparameter between 0 and 1 (often set to 0.9). Briefly explain (you don't need to prove mathematically, just give an intuition) how using $\boldsymbol{m}$ stops the updates from varying as much. Why might this help with learning?

Momentum prevents the gradients from directly influencing/changing the parameters, since it is a moving average of the gradients which smoothens the noise and the gradients of the current iteration are reduced by a factor of 10. Hence there is less fluctuation in the updates due to less noise from the gradients and the current gradient has less magnitude which makes it less influential.

It helps learning because the gradients at a certain iteration might be 0, which causes naïve optimizers like SGD to stop at that local minimum even if the loss at that point is high. With momentum, the update to the parameters are not zero even if the gradient is zero, which prevents the parameters from being stuck at a bad local minimum by having that residual momentum shift the parameters away from the local minimum.

(ii) Adam also uses *adaptive learning rates* by keeping track of $\boldsymbol{v}$, a rolling average of the magnitudes of the gradients:

$$\boldsymbol{m} \leftarrow \beta_1 \boldsymbol{m} + (1-\beta_1)\nabla_{\boldsymbol{\theta}} J_{minibatch}(\boldsymbol{\theta})$$
$$\boldsymbol{v} \leftarrow \beta_2 \boldsymbol{v} + (1-\beta_2)(\nabla_{\boldsymbol{\theta}} J_{minibatch}(\boldsymbol{\theta}) \circ \nabla_{\boldsymbol{\theta}} J_{minibatch}(\boldsymbol{\theta}))$$
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \circ \boldsymbol{m}/\sqrt{\boldsymbol{v}}$$

where $\circ$ and $/$ denote elementwise multiplication and division (so $\boldsymbol{z} \circ \boldsymbol{z}$ is elementwise squaring) and β_2 is a hyperparameter between 0 and 1 (often set to 0.99). Since Adam divides the update by $\sqrt{\boldsymbol{v}}$, which of the model parameters will get larger updates? Why might this help with learning?

Parameters with small gradients will have larger updates, since small gradients mean that they are converging to a possibly bad local minimum. By increasing the update magnitude, it can help these parameters to shift away from the local minimum.

h)

Baseline: Best UAS on dev: 87.94, UAS on test: 88.48

With L2-regularization: Lower

With additional hidden layer of same dimensions: Lower

Q3

(a) (4 points, written) Conventionally, when reporting performance of a language model, we evaluate on *perplexity*, which is defined as:

$$\text{PP}^{(t)}\left(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)}\right) = \frac{1}{\bar{P}(\boldsymbol{x}^{(t+1)}_{\text{pred}} = \boldsymbol{x}^{(t+1)} \mid \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})} = \frac{1}{\sum_{j=1}^{|V|} y_j^{(t)} \cdot \hat{y}_j^{(t)}}$$

i.e. the inverse probability of the correct word, according to the model distribution $\bar{P}$.

(i) (2 points) Show how you can derive perplexity from the cross-entropy loss (*Hint: remember that $\boldsymbol{y}^{(t)}$ is one-hot!*). ***This should be a very short problem - not too perplexing!***

(ii) (1 point) Now use this relationship between perplexity and cross-entropy to show that minimizing the geometric mean perplexity, $\left(\prod_{t=1}^{T} PP(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)})\right)^{1/T}$, is equivalent to minimizing the arithmetic mean cross-entropy loss, $\frac{1}{T}\sum_{t=1}^{T} CE(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)})$, across the training set. (*Hint: for any positive function f, minimizing* $\log(f)$ *is equivalent to minimizing f itself*)

(iii) (1 point) Suppose you have a vocabulary of $|V|$ words and your "language model" works by picking the next word uniformly at random from the vocabulary (mathematically, $\bar{P}(\boldsymbol{x}^{(t+1)} = \boldsymbol{w}_j \mid \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})) = 1/|V|$ for every word $\boldsymbol{w}_j$ in the vocabulary). What would the perplexity $PP(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)})$ for a single word to be? Compute the corresponding cross-entropy loss when $|V| = 10000$.

(i) $$CE\left(\hat{y}^{(t)}, y^{(t)}\right) = -\sum_{j=1}^{|V|} {y_j}^{(t)} \log\left({\hat{y}_j}^{(t)}\right)$$

$$e^{CE\left(\hat{y}^{(t)}, y^{(t)}\right)} = e^{-\sum_{j=1}^{|V|} {y_j}^{(t)} \log\left({\hat{y}_j}^{(t)}\right)} = \frac{1}{e^{\sum_{j=1}^{|V|} {y_j}^{(t)} \log\left({\hat{y}_j}^{(t)}\right)}} = \frac{1}{\sum_{j=1}^{|V|} {y_j}^{(t)} e^{\log\left({\hat{y}_j}^{(t)}\right)}}$$

$$= \frac{1}{\sum_{j=1}^{|V|} {y_j}^{(t)} {\hat{y}_j}^{(t)}} = PP\left(\hat{y}^{(t)}, y^{(t)}\right)$$

(ii) $\min_x \log\big(f(x)\big) = \min_x f(x)$

$$\min_y \left[\prod_{t=1}^{T} PP\big(\hat{y}^{(t)}, y^{(t)}\big)\right]^{1/T} = \min_y \; log\left[\prod_{t=1}^{T} PP\big(\hat{y}^{(t)}, y^{(t)}\big)\right]^{1/T}$$

$$= \min_y \; \frac{1}{T}\log\left(\prod_{t=1}^{T} PP\big(\hat{y}^{(t)}, y^{(t)}\big)\right)$$

$$= \min_y \; \frac{1}{T}\sum_{t=1}^{T} \log(PP\big(\hat{y}^{(t)}, y^{(t)}\big)) = \min_y \; \frac{1}{T}\sum_{t=1}^{T} \mathrm{CE}\big(\hat{y}^{(t)}, y^{(t)}\big)$$

(iii) Let $\hat{y}_j{}^{(t)} = \frac{1}{|V|} \quad \forall j, \; PP\big(\hat{y}^{(t)}, y^{(t)}\big) = \frac{1}{1\left(\frac{1}{|V|}\right)+0} = |V|$

If $|V| = 10000, \; CE\big(\hat{y}^{(t)}, y^{(t)}\big) = \log\Big(PP\big(\hat{y}^{(t)}, y^{(t)}\big)\Big) = \log(10000) = 4$

(b) (7 points, written) Compute the gradients of the loss J (the cross-entropy loss) with respect to the following model parameters at a single point in time t (to save a bit of time, you don't have to compute the gradients with the respect to the biases $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$):

$$\frac{\partial J^{(t)}}{\partial \boldsymbol{U}} \qquad \frac{\partial J^{(t)}}{\partial \boldsymbol{e}^{(t)}} \qquad \left.\frac{\partial J^{(t)}}{\partial \boldsymbol{W}_e}\right|_{(t)} \qquad \left.\frac{\partial J^{(t)}}{\partial \boldsymbol{W}_h}\right|_{(t)}$$

where $|_{(t)}$ denotes the gradient for the appearance of that parameter at time t (in other words, $\boldsymbol{h}^{(t-1)}$ is taken to be fixed, so you don't need to take it's derivative when applying the chain rule and backpropagate to earlier timesteps - you'll do that in part (c)).

Additionally, compute the derivative with respect to the *previous* hidden layer value:

$$\frac{\partial J^{(t)}}{\partial \boldsymbol{h}^{(t-1)}}$$

Hint: you may want to define the following variables:

$$\boldsymbol{z}^{(t)} = \boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e \boldsymbol{e}^{(t)} + \boldsymbol{b}_1$$
$$\boldsymbol{\theta}^{(t)} = \boldsymbol{U}\boldsymbol{h}^{(t)} + \boldsymbol{b}_2$$

and define (and compute the value of) the following error terms:

$$\boldsymbol{\delta}_1^{(t)} = \frac{\partial J}{\partial \boldsymbol{\theta}^{(t)}}$$
$$\boldsymbol{\delta}_2^{(t)} = \frac{\partial J}{\partial \boldsymbol{z}^{(t)}} = \boldsymbol{\delta}_1^{(t)} \frac{\partial \boldsymbol{\theta}^{(t)}}{\partial \boldsymbol{h}^{(t)}} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{z}^{(t)}}$$

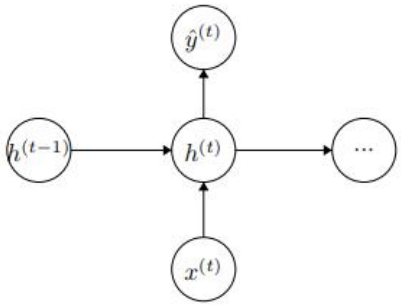
$$\frac{\partial J^{(t)}}{\partial U} = \frac{\partial J^{(t)}}{\partial \theta^{(t)}} \frac{\partial \theta^{(t)}}{\partial U} = \delta_1^{(t)} \big[h^{(t)}\big]^T \qquad \left.\frac{\partial J^{(t)}}{\partial W_e}\right|_t = \frac{\partial J^{(t)}}{\partial z^{(t)}} \left.\frac{\partial z^{(t)}}{\partial W_e}\right|_t = \delta_2^{(t)} \big[e^{(t)}\big]^T$$

$$\frac{\partial J^{(t)}}{\partial e^{(t)}} = \frac{\partial J^{(t)}}{\partial z^{(t)}} \frac{\partial z^{(t)}}{\partial e^{(t)}} = {W_e}^T \delta_2^{(t)} \qquad \left.\frac{\partial J^{(t)}}{\partial W_h}\right|_t = \frac{\partial J^{(t)}}{\partial z^{(t)}} \left.\frac{\partial z^{(t)}}{\partial W_h}\right|_t = \delta_2^{(t)} \big[h^{(t-1)}\big]^T$$

$$\frac{\partial J^{(t)}}{\partial h^{(t-1)}} = \frac{\partial J^{(t)}}{\partial z^{(t)}} \frac{\partial z^{(t)}}{\partial h^{(t-1)}} = {W_h}^T \delta_2^{(t)}$$

Where $\qquad \delta_1^{(t)} = \hat{y}^{(t)} - y^{(t)} \qquad \delta_2^{(t)} = U^T \delta_1^{(t)} \circ \big[\sigma\big(z^{(t)}\big) \circ [1 - \sigma\big(z^{(t)}\big)]\big]$

(c) (7 points, written) Below is a sketch of the network at a single timestep:

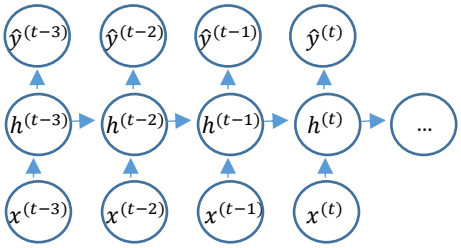


Draw the "unrolled" network for 3 timesteps (your picture should show $h^{(t-3)}$ to $h^{(t)}$. Next, compute the following backpropagation-through-time gradients:

$$\frac{\partial J^{(t)}}{\partial \boldsymbol{e}^{(t-1)}} \qquad \left.\frac{\partial J^{(t)}}{\partial \boldsymbol{W}_e}\right|_{(t-1)} \qquad \left.\frac{\partial J^{(t)}}{\partial \boldsymbol{W}_h}\right|_{(t-1)}$$

where $|_{(t-1)}$ denotes the gradient for the appearance of that parameter at time $(t-1)$. Use the error term $\boldsymbol{\gamma^{(t-1)}} = \frac{\partial J^{(t)}}{\partial \boldsymbol{h}^{(t-1)}}$ to represent the derivative you computed in the previous part (it should show up in three gradients you compute).

We have to compute multiple gradients with respect to parameters like $\boldsymbol{W}_e$ because they are used multiple times in the feed-forward computation. The final gradient for the parameters will be the sum of all their gradients over time (e.g., the gradient $\frac{\partial J^{(t)}}{\partial \boldsymbol{W}_e} = \sum_{i=0}^{t} \frac{\partial J^{(t)}}{\partial \boldsymbol{W}_e}|_{(i)}$).



Let $\gamma^{(t-1)} = \frac{\partial J^{(t)}}{\partial h^{(t-1)}} = {W_h}^T \delta_2^{(t)}$

$$\frac{\partial J}{\partial e^{(t-1)}} = \frac{\partial J^{(t)}}{\partial h^{(t-1)}} \frac{\partial h^{(t-1)}}{\partial z^{(t-1)}} \frac{\partial h^{(t-1)}}{\partial e^{(t-1)}} = {W_e}^T (\gamma^{(t-1)} \circ \sigma'(z^{(t-1)}))$$

$$\left.\frac{\partial J}{\partial W_e}\right|_{t-1} = \frac{\partial J^{(t)}}{\partial h^{(t-1)}} \frac{\partial h^{(t-1)}}{\partial z^{(t-1)}} \frac{\partial h^{(t-1)}}{\partial W_e} = (\gamma^{(t-1)} \circ \sigma'(z^{(t-1)}))[e^{(t-1)}]^T$$

$$\left.\frac{\partial J}{\partial W_h}\right|_{t-1} = \frac{\partial J^{(t)}}{\partial h^{(t-1)}} \frac{\partial h^{(t-1)}}{\partial z^{(t-1)}} \frac{\partial h^{(t-1)}}{\partial W_h} = (\gamma^{(t-1)} \circ \sigma'(z^{(t-1)}))[h^{(t-2)}]^T$$

(d) (3 points, written) Given $\boldsymbol{h}^{(t-1)}$, how many operations are required to perform backpropagation for a single timestep (i.e., compute the terms you found in part (b)). Express your answer in big-O notation in terms of the dimensions d, D_h and $|V|$ (Equation 1). Don't worry about the gradients you didn't compute (with respect to $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$), they actually don't change the answer!

(*Hint: You only have to worry about matrix multiplications, the other operations do not change the big-O runtime. Multiplying a vector by an n by m matrix* takes $O(nm)$ operations.)

$$\frac{\partial J^{(t)}}{\partial U} = \delta_1^{(t)}\left[h^{(t)}\right]^T,\ \delta_1^{(t)} = \hat{y}^{(t)} - y^{(t)} \to O(|V|),\ \left[h^{(t)}\right]^T \to O(D_h)$$

$$\therefore \frac{\partial J^{(t)}}{\partial U} \to O(|V| \cdot D_h)$$

$$\delta_2^{(t)} = U^T\delta_1^{(t)} \circ \left[\sigma\left(z^{(t)}\right) \circ \left[1 - \sigma\left(z^{(t)}\right)\right]\right] = U^T\delta_1^{(t)} \circ \sigma'\left(z^{(t)}\right)$$

$$U^T\delta_1^{(t)} \to O(|V| \cdot D_h \cdot 1),\ \sigma\left(z^{(t)}\right) \to 3O(D_h) \to O(D_h), \sigma'\left(z^{(t)}\right) \to O({D_h}^2)$$

$$\left[U^T\delta_1^{(t)}\right] \circ \sigma'\left(z^{(t)}\right) \to O({D_h}^2)$$

$$\therefore \delta_2^{(t)} \to O(|V| \cdot D_h), \text{Assuming } |V| > D_h$$

$$\frac{\partial J^{(t)}}{\partial e^{(t)}} = {W_e}^T\delta_2^{(t)} \to O(d \cdot D_h) \therefore \frac{\partial J^{(t)}}{\partial e^{(t)}} \to O(|V| \cdot D_h), \text{Assuming } |V| > d$$

$$\frac{\partial J^{(t)}}{\partial h^{(t-1)}} = {W_h}^T\delta_2^{(t)} \to O({D_h}^2) \therefore \frac{\partial J^{(t)}}{\partial h^{(t-1)}} \to O(|V| \cdot D_h)$$

$$\left.\frac{\partial J^{(t)}}{\partial W_e}\right|_t = \delta_2^{(t)}\left[e^{(t)}\right]^T \to O(d \cdot D_h)\ \therefore \left.\frac{\partial J^{(t)}}{\partial W_e}\right|_t \to O(|V| \cdot D_h)$$

$$\left.\frac{\partial J^{(t)}}{\partial W_h}\right|_t = \delta_2^{(t)}\left[h^{(t-1)}\right]^T \to O\left({D_h}^2\right) \therefore \left.\frac{\partial J^{(t)}}{\partial W_h}\right|_t \to O(|V| \cdot D_h)$$

The biases b_1, b_2 do not change the complexity since they are $O(D_h), O(|V|)$ respectively. We drop lower order complexities.

Solution: $O\left(|V| \cdot D_h + d \cdot D_h + {D_h}^2\right)$ for the general case where the values of $|V|$, D_h and d are unknown.

(e) (3 points, written) Now suppose you have a sequence of T words. How many operations are required to compute the gradient of the loss with respect to the model parameters across the entire sequence ($\sum_{t=1}^{T} J^{(t)}(\theta)$)? Assume we backpropagate through time all the way to $t = 0$ for each word.

Hint: Look at the computational graph you drew in part (c). Will we have to pass an error signal (upstream gradient) into $h^{(0)}$ T times (once for the loss from every word), or can we be more efficient that that? Therefore, how many times will we have to do the single timestep (your answer to (d)) computations?

Computing $\frac{\partial J^{(t)}}{\partial *}$ requires T steps for each t, so it is $O\left(T\left[|V| \cdot D_h + d \cdot D_h + {D_h}^2\right]\right)$. Only one backward pass is made since we sum the gradients $\frac{\partial J^{(t)}}{\partial *}$ up, else it would be $O(T^2[\ldots])$. T is not a constant so we cannot drop it.

(f) (1 point, written) Which term in your big-O expressions is likely the largest? Which layer in the RNN is that term from?

$O(|V| \cdot D_h)$ is largest. Due to $\delta_2^{(t)}$.